

Ex. No.: 01

DATE: 07/08/24

**STUDY THE INSTRUCTION SET OF 8051 MICROCONTROLLER
AND EXECUTE SIMPLE INSTRUCTIONS**

Ex. No.: 1a

8 - BIT ADDITION

AIM:

To write an assembly language program to perform addition of two 8-bit numbers using 8051 Microcontroller.

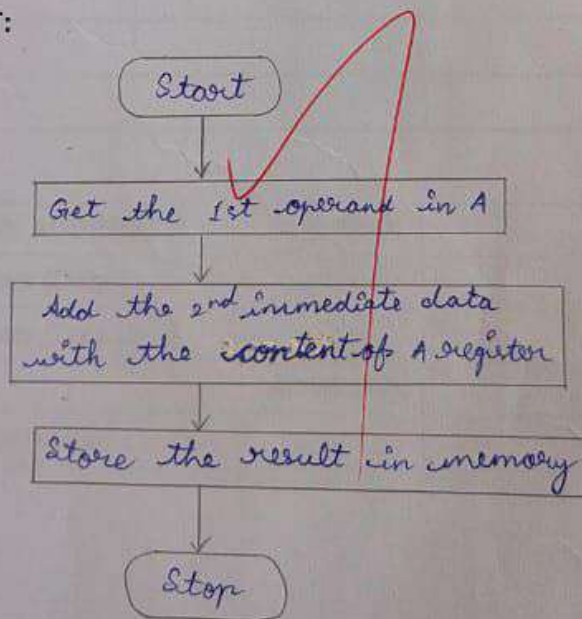
APPARATUS REQUIRED:

1. 8051-Microcontroller Kit.
2. Keyboard.

ALGORITHM:

1. Load the first data in accumulator.
2. Load the second data in B register.
3. Add the content of accumulator with the content of B register.
4. Run the program.
5. The result will be stored in accumulator.

FLOW CHART:



PROGRAM:

```
MOV A,#62H
MOV B,#34H
ADD A,B
END
```

Address	Mnemonics	Operands	Hex. Code	Comments
4100	MOV	A, #34	74	Get the 1st operand '34' in the Accumulator register
4101			34	
4102	ADD	A, #62	24	Add the 2nd operand '62' with the content of Accumulator
4103			62	
4104	MOV	DPTR, #4500	90	Move the memory address 4500 into the Data Pointer register.
4105			45	
4106			00	
4107	MOVX	@DPTR, A	F0	Store the result into memory address specified in the Data Pointer
4108	SJMP	4108	80	Stop the program
4109			FE	

MEMORY ADDRESS	ADDRESS	INPUT
A register (Data 1)	4101	34
B register (Data 2)	4103	62
	ADDRESS	OUTPUT
Accumulator	4500	96

Ex. No.: 1b

07/08/24

8 - BIT SUBTRACTION

AIM:

To write an assembly language program to perform subtraction of two 8-bit numbers using Microcontroller.

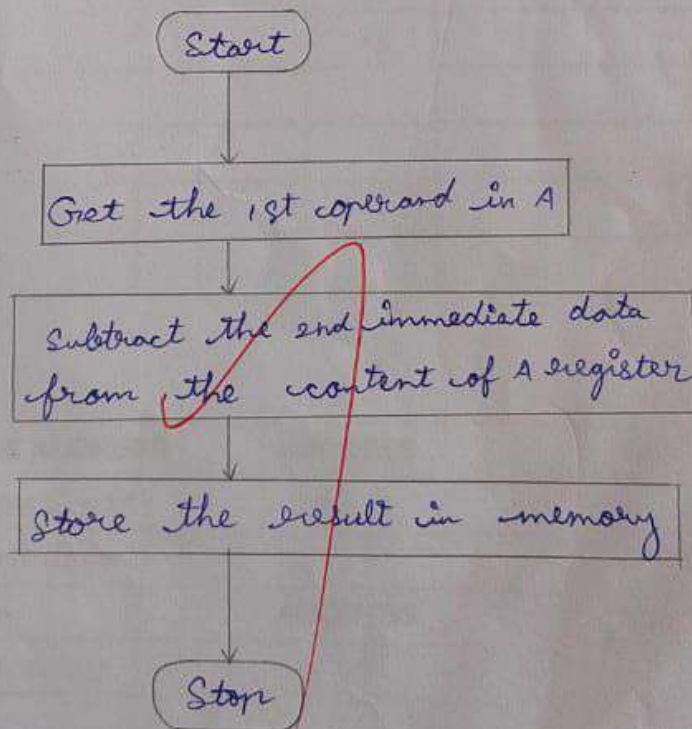
APPARATUS REQUIRED:

1. 8051-Microcontroller Kit.
2. Keyboard.

ALGORITHM:

1. Load the first data in accumulator.
2. Load the second data in B register.
3. Subtract the content of B register from accumulator.
4. Run the program.
5. The result will be stored in accumulator.

FLOW CHART:



PROGRAM:

```
MOV A,#62H
MOV B,#34H
SUBB A,B
END
```

Address	Mnemonics	Operands	Hex. Code	Comments
4100	MOV	A, #12	74	Get the 1st operand '12' in the Accumulator register
4101			12	
4102	SUBB	A, #08	94	Subtract the 2nd operand '08' from the content of Accumulator
4103			08	
4104	MOV	DPTR, #4500	90	Move the memory address 4500 into the Data Pointer register
4105			45	
4106			00	
4107	MOVX	@DPTR, A	F0	Store the result into memory address specified in the Data Pointer
4108	SJMP	4108	80	Stop the program
4109			FE	

MEMORY ADDRESS	ADDRESS	INPUT
A register (Data 1)	4101	12
B register (Data 2)	4103	08
	ADDRESS	OUTPUT
Accumulator	4500	0A

Ex. No.: 1c

07/08/24

8 - BIT MULTIPLICATION

AIM:

To write an assembly language program to perform multiplication of two 8-bit numbers using Microcontroller.

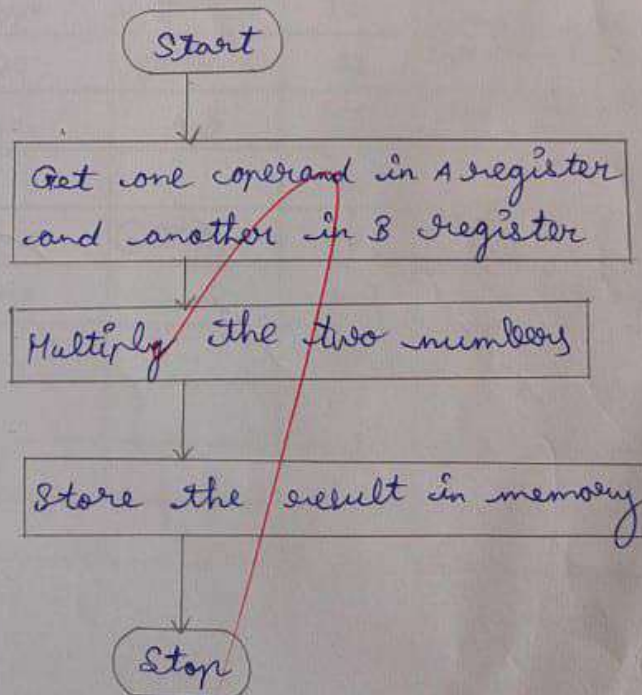
APPARATUS REQUIRED:

1. 8051-Microcontroller Kit.
2. Keyboard.

ALGORITHM:

1. Load the first data in accumulator.
2. Load the second data in B register.
3. Multiply the content of accumulator with the content of B register.
4. Run the program.
5. The lower byte of result will be stored in accumulator and higher byte of result will be stored in B register.

FLOW CHART:



PROGRAM:

MOV A,#62H

MOV B,#34H

MUL A,B

END

Address	Mnemonics	Operands	Hex. Code	Comments
4100	MOV	A, #04	74	Load the 1st operand in the Accumulator register
4101			04	
4102	MOV	B, #02	75	Load the 2nd operand in the B register
4103			F0	
4104			02	
4105	MUL	AB	A4	Multiply the two numbers
4106	MOV	DPTR, #4500	90	Move the memory address 4500 into the data pointer register
4107			00	
4108			45	
4109	MOVX	@DPTR, A	F0	Store the lower byte of result into memory address specified in the Data Pointer
410A	INC	DPTR	A3	Increment the content of the Data Pointer
410B	MOV	A, B	E5	Move the content of B into A register
410C			F0	
410D	MOVX	@DPTR, A	F0	Store the higher byte of result into memory address specified in the Data Pointer
410E	SJMP	410E	80	Stop the program
410F			FE	

MEMORY ADDRESS	ADDRESS	INPUT
A register (Data 1)	4101	04
B register (Data 2)	4103	02
	ADDRESS	OUTPUT
Accumulator	4500	08

Ex. No.: 1d

07/08/24

8 - BIT DIVISION

AIM:

To write an assembly language program to perform division of two 8-bit numbers using Microcontroller.

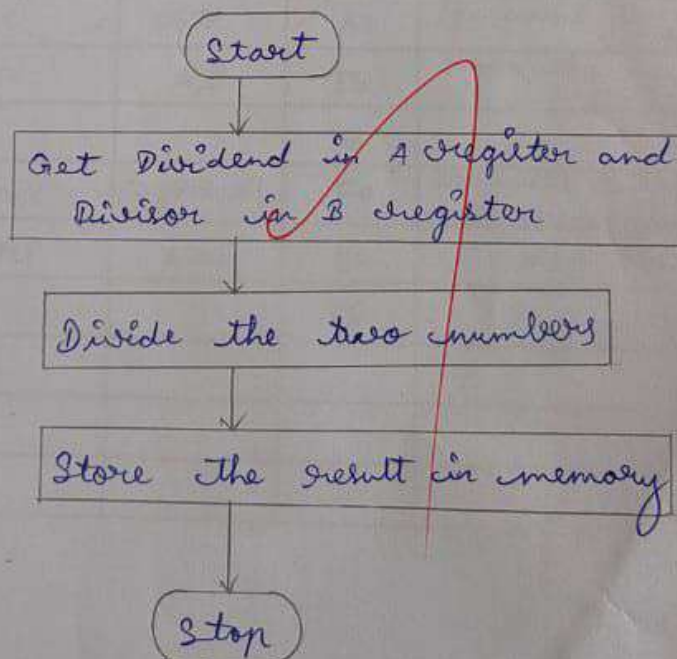
APPARATUS REQUIRED:

1. 8051-Microcontroller Kit.
2. Keyboard.

ALGORITHM:

1. Load the first data in accumulator.
2. Load the second data in B register.
3. Divide the content of accumulator by the content of B register.
4. Run the program.
5. The quotient will be stored in accumulator and remainder will be stored in B register.

FLOW CHART:



PROGRAM:

```
MOV A,#62H
MOV B,#34H
DIV A,B
END
```

Address	Mnemonics	Operands	Hex. Code	Comments
4100	MOV	A,#09	74	Load Accumulator with dividend
4101			09	
4102	MOV	B,#02	75	Load the register B with Divisor
4103			F0	
4104			02	
4105	DIV	AB	84	Divide the two numbers
4106	MOV	DPTR,#4500	90	Move the memory address 4500 into the Data Pointer register.
4107			45	
4108			00	
4109	MOVB	@DPTR,A	F0	Store the Quotient into memory address specified in the Data Pointer
410A	INC	DPTR	A3	Increment the content of the Data Pointer
410B	MOV	A,B	E5	Move the content of B into A register
410C			F0	
410D	MOVB	@DPTR,A	F0	Store the Remainder into memory address specified in the Data Pointer
410E	SJMP	410E	80	Stop the program
410F			FE	

MEMORY ADDRESS	ADDRESS	INPUT
A register (Data 1)	4101	09
B register (Data 2)	4104	02
	ADDRESS	OUTPUT
Accumulator (quotient)	4500	04
B register (remainder)	4501	01

RESULT:

Thus the addition, subtraction, multiplication & division of two 8 bit numbers was performed using 8051 Microcontroller kit and verified the result.

Ex. No.: 3a

KEYPAD AND DISPLAY INTERFACING MICROCONTROLLER

AIM:

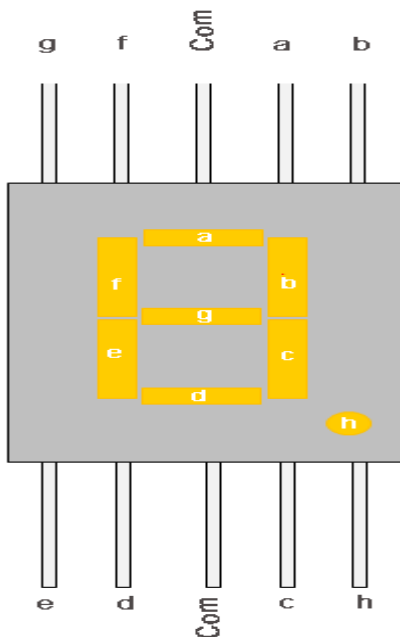
To design and compile the source code for any embedded system application using Keil uVision4 compiler.

APPARATUS REQUIRED:

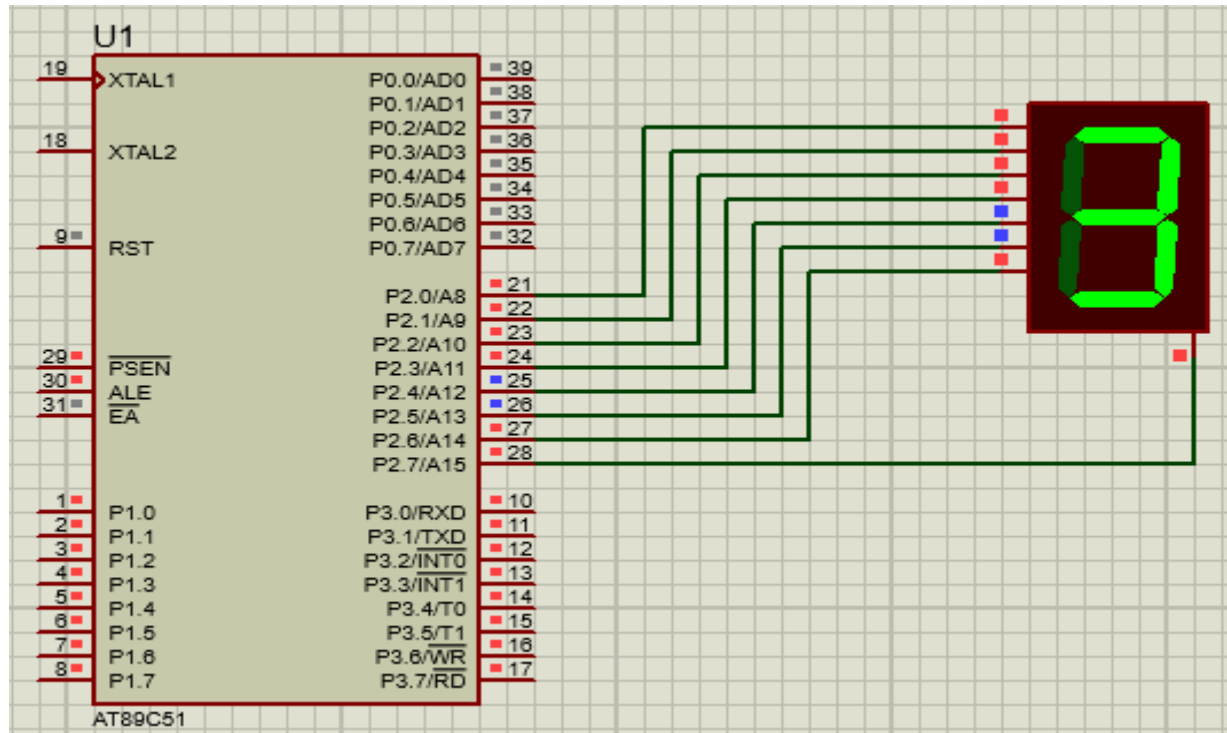
Keil uVision4 compiler and Proteus.

DEFINITIONS /THEORY:

Seven segment displays are used to indicate numerical information. Seven segments display can display digits from 0 to 9 and even we can display few characters like A, b, C, H, E, e, F, etc. These are very popular and have many more applications. So, in this project, I'll show you how a 7 Segment Display works by interfacing 7 Segment Display to 8051 Microcontroller.



Circuit Diagram



Lab Work:

PROGRAM

```
ORG 00H
START:MOV R1,#10
MOV DPTR,#400H
BACK:CLR A
MOVC A,@A+DPTR
MOV P2,A
ACALL DELAY
INC DPTR
DJNZ R1,BACK
SJMP START
ORG 400H
DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
```



```
DELAY:MOV R2,#08H
UP2:MOV R4,#0FFH
UP1:MOV R3,#0FFH
HERE:DJNZ R3,HERE
      DJNZ R4,UP1
      DJNZ R2,UP2
      RET
      END
```

OUTPUT:

Inference:

Result

The design and compile the source code for any embedded system application of Seven Segment Display using 8051 is done using Keil uVision4 compiler.

Ex. No.: 3b**KEYPAD INTERFACING MICROCONTROLLER****AIM:**

To design and compile the source code for any embedded system application using Keil uVision4 compiler.

APPARATUS REQUIRED:

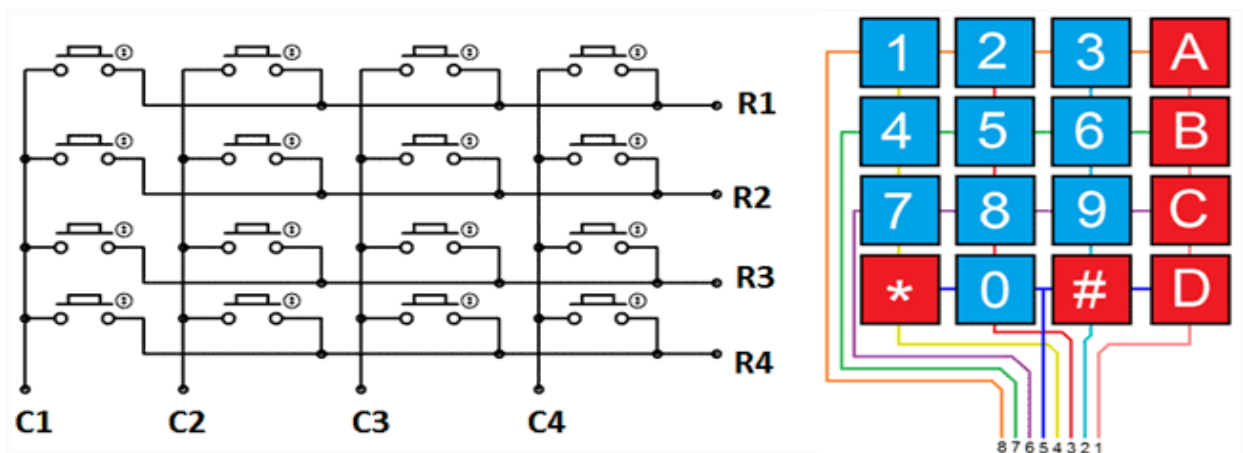
Keil uVision4 compiler and Proteus.

DEFINITIONS /THEORY:

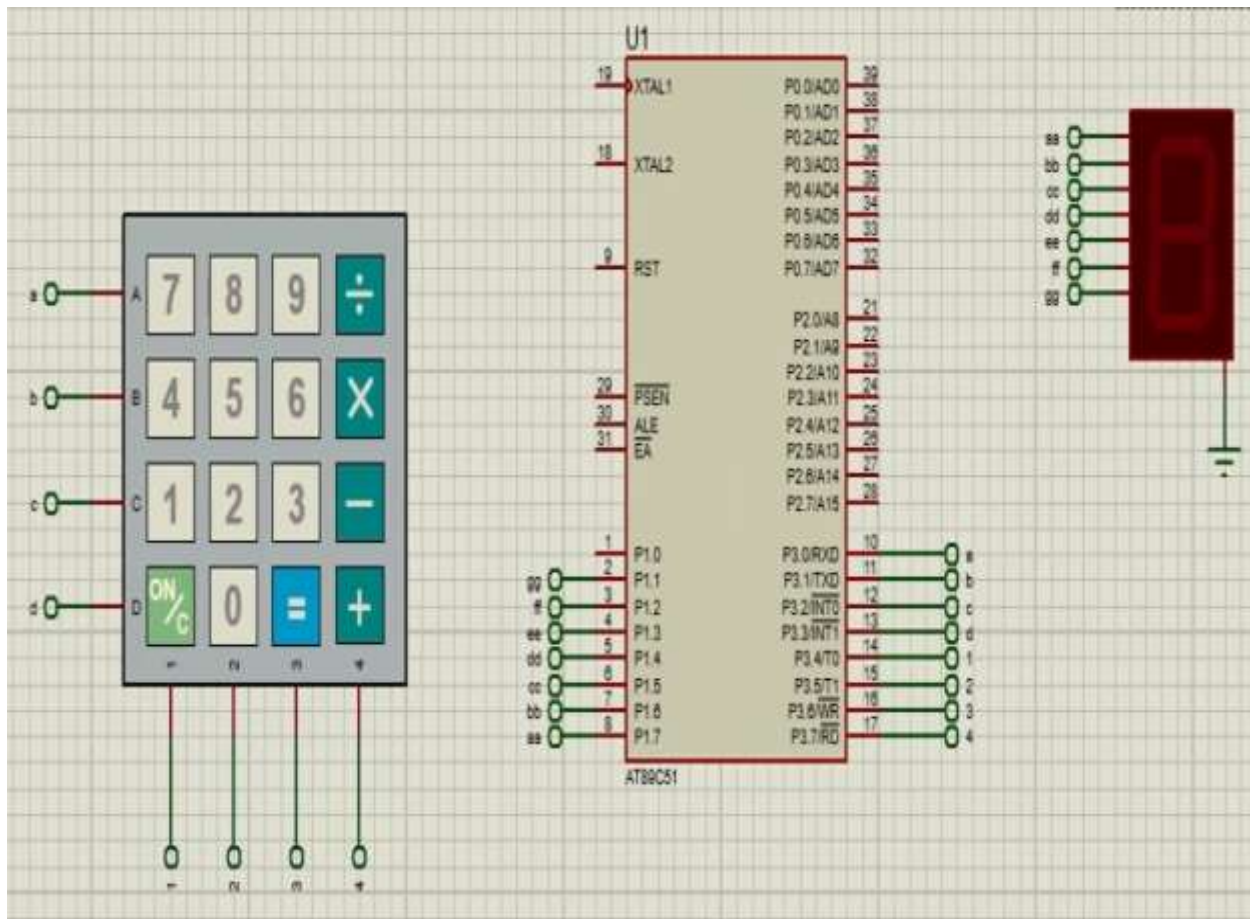
Keypads are widely used input devices being used in various electronics and embedded projects. They are used to take inputs in the form of numbers and alphabets, and feed the same into system for further processing. In this tutorial we are going to interface a 4x4 matrix keypad with 8051 microcontroller.

4X4 Matrix Keypad

Before we interface the keypad with microcontroller, first we need to understand how it works. Matrix keypad consists of set of Push buttons, which are interconnected. Like in our case we are using 4X4 matrix keypad, in which there are 4 push buttons in each of four rows. And the terminals of the push buttons are connected according to diagram. In first row, one terminal of all the 4 push buttons are connected together and another terminal of 4 push buttons are representing each of 4 columns, same goes for each row. So, we are getting 8 terminals to connect with a microcontroller.



Circuit Diagram



Lab Work:**PROGRAM**

```
ORG 00H
MOV DPTR,#look_up_table
MOV A,#0FFH
MOV P1,#00000000B
reverse:MOV P3,#0FFH
CLR P3.0
JB P3.4,next_find_1
MOV A,#0D
ACALL disp_000
next_find_1:JB P3.5,next_find_2
MOV A,#1D
ACALL disp_000
next_find_2:JB P3.6,next_find_3
MOV A,#2D
ACALL disp_000
next_find_3:JB P3.7,next_find_4
MOV A,#3D
ACALL disp_000
next_find_4:SETB P3.0
CLR P3.1
JB P3.4,next_find_5
MOV A,#4D
ACALL disp_000
next_find_5:JB P3.5,next_find_6
MOV A,#5D
ACALL disp_000
next_find_6:JB P3.5,next_find_7
MOV A,#6D
ACALL disp_000
next_find_7:JB P3.5,next_find_8
MOV A,#7D
ACALL disp_000
next_find_8:SETB P3.1
CLR P3.2
JB P3.4,NEXT9
MOV A,#8D
ACALL disp_000
NEXT9:JB P3.5,next_find_10
MOV A,#9D
```

```
ACALL disp_000
next_find_10:JB P3.6,next_find_11
MOV A,#10D
ACALL disp_000
next_find_11:JB P3.7,next_find_12
MOV A,#11D
ACALL disp_000
next_find_12:SETB P3.2
CLR P3.3
JB P3.4, next_find_13
MOV A,#12D
ACALL disp_000
next_find_13:JB P3.5,next_find_14
MOV A,#13D
ACALL disp_000
next_find_14:JB P3.6,next_find_15
MOV A,#14D
ACALL disp_000
next_find_15:JB P3.7,reverse
MOV A,#15D
ACALL disp_000
LJMP reverse
disp_000:MOVC A,@A+DPTR
MOV P1,A
```

```
RET
look_up_table:
DB 11100000B
DB 11111110B
DB 11110110B
DB 10011100B
DB 01100110B
DB 10110110B
DB 10111110B
DB 00111110B
DB 01100000B
DB 11011010B
DB 11110010B
DB 11101110B
DB 10011110B
DB 11111100B
```


DB 10001110B
DB 01111010B
END

OUTPUT:

Inference:

Result

The design and compile the source code for any embedded system application of Interfacing Matrix Keyboard using 8051 is done using Keil uVision4 compiler.

Ex. No.: 2a

EMBEDDED C CODE TO BLINKING LEDS

AIM:

To design and compile the Embedded C source code to control LEDs using Keil uVision4 compiler.

APPARATUS REQUIRED:

Keil uVision4 compiler and Proteus.

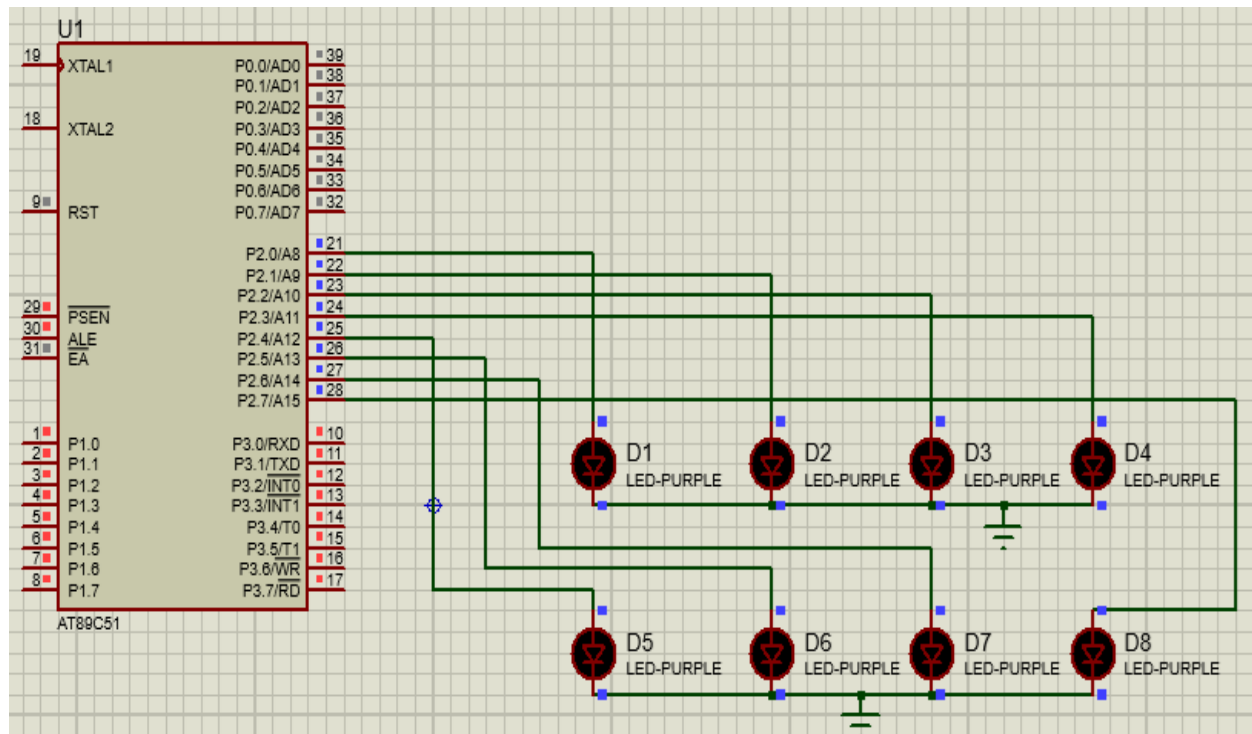
DEFINITIONS /THEORY:

Light Emitting Diodes (LEDs) are simple and most commonly used electronic components to display the digital signal states. The LED emits light when current is passed through it. It could blow up if we pass more current, hence we put a current limiting resistor. Usually, 220, 470 and 1K ohm resistors are commonly used as limiting resistors. You can use any of these depending on the required brightness. Lets, start blinking with LEDs and then generate the different patterns using the available LEDs .

The basic and important feature of any controllers is the number of GPIO's available for connecting the peripherals. 8051 has 32-gpio's grouped into 4-Ports namely P0-P3 as shown in the below table.

PORT	Number of Pins	Alternative Function
P0	8 (P0.0-P0.7)	AD0-AD7 (Address and Data bus)
P1	8 (P1.0-P1.7)	None
P2	8 (P2.0-P2.7)	A8-A15 (Higher Address Bus)
P3	8 (P3.0-P3.7)	UART, Interrupts, (T0/T1)Counters

Circuit Diagram



Lab Work:

PROGRAM

```
#include<reg51.h>
void delay()
{
    int t;
    for(t=0;t<32000;t++);
}
void main()
{
    while(1)
    {
        P2=0xff;
        delay();
        P2=0x00;
        delay();
    }
}
```

OUTPUT:

Inference:

Result

Thus the compilation of the Embedded C source code to control LEDs using 8051 is done using Keil uVision compiler.

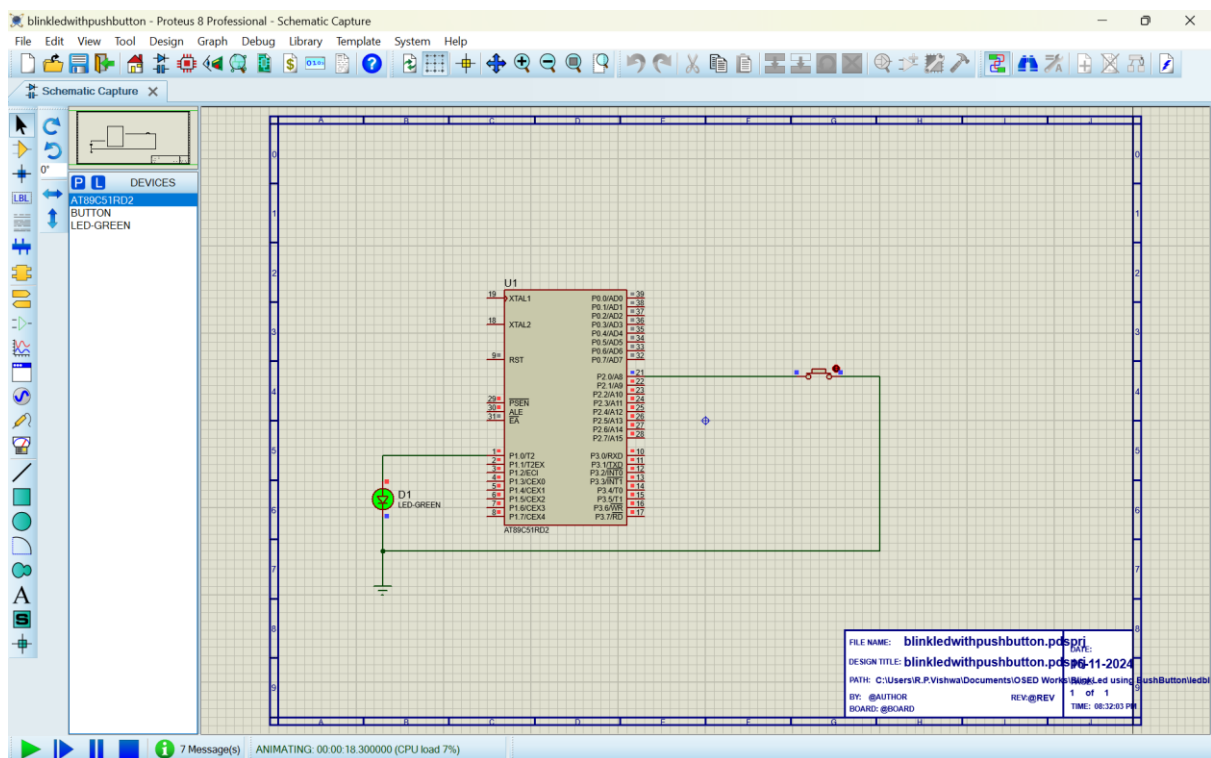
Ex.No—4

Blink Led with Push Button

AIM:

Apparatus Required:

Circuit:



Coding Part:

```
#include<reg51.h>
```

```
sbit led=P1^0;
```

```
sbit sw=P2^0;
```

```
void main(){
```

```
    led=0;
```

```
    sw=1;
```

```
    if(sw==0){
```

```
        led=1;
```

```
        while(sw==0);
```

```
    }
```

```
}
```

Ex. No.: 5a

STUDY BASIC AND USER STATUS LINUX COMMANDS

AIM:

To study the basics of linux commands.

DEFINITIONS /THEORY:

Inference:

Linux commands are a type of Unix command or shell procedure. They are the basic tools used to interact with Linux on an individual level. Linux commands are used to perform a variety of tasks, including displaying information about files and directories.

Linux operating system is used on servers, desktops, and maybe even your smartphone. It has a lot of command line tools that can be used for virtually everything on the system.

Linux Commands:

1. **ls** - The most frequently used command in Linux to list directories
2. **pwd** - Print working directory command in Linux
3. **cd** - Linux command to navigate through directories
4. **mkdir** - Command used to create directories in Linux
5. **mv** - Move or rename files in Linux
6. **cp** - Similar usage as mv but for copying files in Linux
7. **rm** - Delete files or directories
8. **touch** - Create blank/empty files
9. **ln** - Create symbolic links (shortcuts) to other files
10. **clear** - Clear the terminal display
11. **cat** - Display file contents on the terminal
12. **echo** - Print any text that follows the command
13. **less** - Linux command to display paged outputs in the terminal
14. **man** - Access manual pages for all Linux commands
15. **uname** - Linux command to get basic information about the OS
16. **whoami** - Get the active username
17. **tar** - Command to extract and compress files in linux
18. **grep** - Search for a string within an output
19. **head** - Return the specified number of lines from the top
20. **tail** - Return the specified number of lines from the bottom
21. **diff** - Find the difference between two files
22. **cmp** - Allows you to check if two files are identical
23. **comm** - Combines the functionality of diff and cmp
24. **sort** - Linux command to sort the content of a file while outputting
25. **export** - Export environment variables in Linux
26. **zip** - Zip files in Linux
27. **unzip** - Unzip files in Linux
28. **ssh** - Secure Shell command in Linux
29. **service** - Linux command to start and stop services

30. **ps** - Display active processes
31. **kill and killall** - Kill active processes by process ID or name
32. **df** - Display disk filesystem information
33. **mount** - Mount file systems in Linux
34. **chmod** - Command to change file permissions
35. **chown** - Command for granting ownership of files or folders
36. **ifconfig** - Display network interfaces and IP addresses
37. **traceroute** - Trace all the network hops to reach the destination
38. **wget** - Direct download files from the internet
39. **ufw** - Firewall command
40. **iptables** - Base firewall for all other firewall utilities to interface with
41. **apt, pacman, yum, rpm** - Package managers depending on the distribution
42. **sudo** - Command to escalate privileges in Linux
43. **cal** - View a command-line calendar
44. **alias** - Create custom shortcuts for your regularly used commands
45. **dd** - Majorly used for creating bootable USB sticks
46. **whereis** - Locate the binary, source, and manual pages for a command
47. **whatis** - Find what a command is used for
48. **top** - View active processes live with their system usage
49. **useradd and usermod** - Add a new user or change existing user data
50. **passwd** - Create or update passwords for existing users

1. pwd command

The **pwd** command (**print working directory**) is a shell builtin command that prints the current location. The output shows an absolute directory path, starting with the root directory (**/**).

The general syntax is:

```
pwd <options>
```

To see how the command works, run the following in the terminal:

```
pwd
```



```
kb@phoenixNAP:~$ pwd
/home/kb
kb@phoenixNAP:~$
```

The output prints the current location in the **/home/<username>** format.

2. ls command

The **ls** command (**list**) prints a list of the current directory's contents. Run the following:

```
ls
```

```
kb@phoenixNAP:~$ ls
Desktop  Downloads  Pictures  snap      Videos
Documents Music      Public   Templates
```

Additional options provide flexibility with the display output. Typical usage includes combining the following options:

- Show as a list:

```
ls -l
```

- Show as a list and include hidden files:

```
ls -la
```

- Show sizes in a human-readable format:

```
ls -lah
```

3. cd command

The `cd` command (change directory) is a shell builtin command for changing the current working directory:

```
cd <directory>
```

For example, to move to the *Document* directory, run:

```
cd Documents
```

```
kb@phoenixNAP:~$ cd Documents
kb@phoenixNAP:~/Documents$
```

The working directory changes in the terminal interface. In a non-default interface, use the **pwd** command to check the current directory.

Use **cd** without any parameters to return to the home directory (~).

4. cat command

The cat command (concatenate) displays the contents of a file in the terminal (standard output or stdout). To use the command, provide a file name from the current directory:

```
cat <filename>
```

```
kb@phoenixNAP:~$ cat file.txt
Hello world!
kb@phoenixNAP:~$
```

Alternatively, provide a path to the file along with the file name:

```
cat <path>/<filename>
```

The command can also:

- Display contents of multiple files:

```
cat <file 1> <file 2>
```

- Create new files:

```
cat ><filename>
```

Add contents to the file and press **CTRL+D** to exit.

- Display line numbers:

```
cat -n <filename>
```

5. touch command

The primary purpose of the touch command is to modify an existing file's timestamp. To use the command, run:

```
touch <filename>
```

```
kb@phoenixNAP:~$ touch new_file.txt
kb@phoenixNAP:~$ ls
Desktop  Downloads  Music      Pictures  Templates
Documents file.txt   new_file.txt Public    Videos
kb@phoenixNAP:~$
```

The command creates an empty file if it does not exist. Due to this effect, **touch** is also a quick way to make a new file (or a batch of files).

6. cp command

The main way to copy files and directories in Linux is through the cp command (**copy**). Try the command with:

```
cp <source file> <target file>
```

```
kb@phoenixNAP:~$ cp file.txt file_copy.txt
kb@phoenixNAP:~$ ls
Desktop  Downloads  file.txt  Pictures  Templates
Documents file_copy.txt Music    Public    Videos
kb@phoenixNAP:~$
```

The source and target files must have different names since the command copies in the same directory. Provide a path before the file name to copy to another location.

7. mv command

Use the mv command (**move**) to move files or directories from one location to another. For example, to move a file from the current directory to *~/Documents*, run:

```
mv <filename> ~/Documents/<filename>
```

```
kb@phoenixNAP:~$ mv file.txt ~/Documents/file.txt
kb@phoenixNAP:~$ ls ~/Documents
file.txt
kb@phoenixNAP:~$
```

8. mkdir command

The mkdir command (**make directory**) creates a new directory in the provided location. Use the command in the following format:


```
mkdir <directory name>
```

```
kb@phoenixNAP:~$ mkdir New_directory
kb@phoenixNAP:~$ ls
Desktop  Downloads  New_directory  Public  Videos
Documents Music      Pictures      Templates
kb@phoenixNAP:~$
```

Provide a path to create a directory in the given location, or use a space or comma-separated list to create multiple directories simultaneously.

9. rmdir command

Use the rmdir command (**remove directory**) to delete an empty directory. For example:

```
rmdir <directory name>
```

```
kb@phoenixNAP:~$ ls
Desktop  Downloads  New_directory  Public  Videos
Documents Music      Pictures      Templates
kb@phoenixNAP:~$ rmdir New_directory
kb@phoenixNAP:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
kb@phoenixNAP:~$
```

If the directory is not empty, the command fails.

10. rm command

The **rm** command (**remove**) deletes files or directories. To use the command for non-empty directories, add the **-r** tag:

```
rm -r <file or directory>
```

```
kb@phoenixNAP:~$ ls Documents
file.txt
kb@phoenixNAP:~$ rm -r Documents
kb@phoenixNAP:~$ ls
Desktop  Downloads  Music  Pictures  Public  Templates  Videos
kb@phoenixNAP:~$
```

Unlike the **rmdir** command, **rm** also removes all the contents from the directory.

Note: Removing some directories in Linux is dangerous. Make sure you know what you're removing before running a dangerous Linux terminal command.

11. locate command

The locate command is a simple Linux tool for finding a file. The command checks a file database on a system to perform the search quickly. However, the result is sometimes inaccurate if the database is not updated.

To use the command, install locate and try the following example:

```
locate <filename>
```

```
kb@phoenixNAP:~$ locate file.txt
/home/kb/Documents/file.txt
/usr/share/doc/alsa-base/driver/Procfile.txt.gz
kb@phoenixNAP:~$
```

The output prints the file's location path. The matching is unclear and outputs any file that contains the file name.

12. find command

Use the find command to perform a thorough search on the system. Add the **-name** tag to search for a file or directory by name:

```
find -name <file or directory>
```

```
kb@phoenixNAP:~$ find -name file.txt
./Documents/file.txt
kb@phoenixNAP:~$
```

The output prints the file's path and performs an exact match. Use additional options to control the search further.

13. grep command

The grep command (**g**lobal **r**egular **e**xpression **p**rint) enables searching through text in a file or a standard output. The basic syntax is:

```
grep <search string> <filename>
```

```
kb@phoenixNAP:~$ grep world Documents/file.txt
Hello world!
kb@phoenixNAP:~$
```

The output highlights all matches. Advanced commands include using grep for multiple strings or writing grep regex statements.

14. sudo command

The sudo command (**superuser do**) elevates a user's permissions to administrator or root. Commands that change system configuration require elevated privilege.

Add **sudo** as a prefix to any command that requires elevated privileges:

```
sudo <command>
```

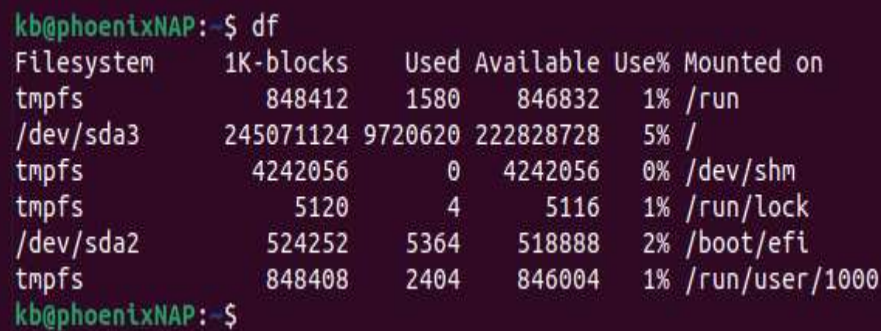
Use the command with caution to avoid making accidental changes permanent.

Note: Learn more about Linux file permissions.

15. df command

The **df** command (**disk free**) is used to check available disk space on the file system. To see how **df** works, run the following:

```
df
```



```
kb@phoenixNAP:~$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
tmpfs            848412      1580    846832   1% /run
/dev/sda3       245071124  9720620  222828728   5% /
tmpfs            4242056        0    4242056   0% /dev/shm
tmpfs            5120         4        5116   1% /run/lock
/dev/sda2        524252     5364    518888   2% /boot/efi
tmpfs            848408     2404    846004   1% /run/user/1000
kb@phoenixNAP:~$
```

The output shows the amount of space used by different drives. Add the **-h** tag to make the output in human-readable format (kilobytes, megabytes, and gigabytes).

16. du command

The **du** (**disk usage**) command helps show how much space a file or directory takes up. Run the command without any parameters:

```
du
```

```
kb@phoenixNAP:~$ du
8      ./Documents
4      ./Public
4      ./Downloads/firefox.tmp/Temp-ba964148-d1ac-4718-ab72-33eda062843d
508    ./Downloads/firefox.tmp
512    ./Downloads
4      ./Videos
4      ./Templates
4      ./Pictures
4      ./local/share/gnome-settings-daemon
```

The output shows the amount of space used by files and directories in the current directory. The size displays in blocks, and adding the **-h** tag changes the measure to human-readable format.

17. head command

Use the head command to truncate long outputs. The command can truncate files, for example:

```
head <filename>
```

Alternatively, pipe **head** to a command with a long output:

```
<command> | head
```

For example, to see the first ten lines of the **du** command, run:

```
du | head
```

```
kb@phoenixNAP:~$ du | head
8      ./Documents
4      ./Public
4      ./Downloads/firefox.tmp/Temp-ba964148-d1ac-4718-ab72-33eda062843d
508    ./Downloads/firefox.tmp
512    ./Downloads
4      ./Videos
4      ./Templates
4      ./Pictures
4      ./local/share/gnome-settings-daemon
4      ./local/share/icc
kb@phoenixNAP:~$
```

The output shows the first ten lines instead of everything.

18. tail command

The Linux tail command does the opposite of **head**. Use the command to show the last ten lines of a file:

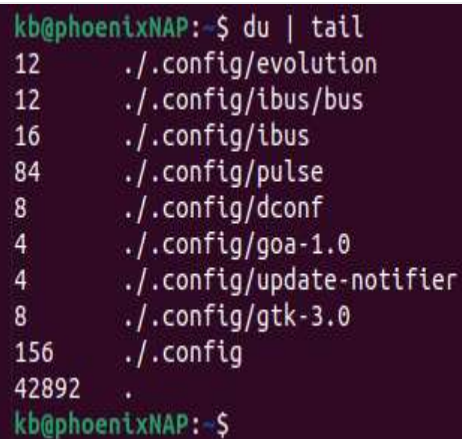
```
tail <filename>
```

Or pipe **tail** to a command with a long output:

```
<command> | tail
```

For example, use **tail** to see the last ten lines of the **du** command:

```
du | tail
```

A terminal window with a dark purple background. The prompt is 'kb@phoenixNAP:~\$'. The command 'du | tail' has been entered. The output shows the size of the last ten directories in the current path, sorted by size. The output is: 12 ./.config/evolution, 12 ./.config/ibus/bus, 16 ./.config/ibus, 84 ./.config/pulse, 8 ./.config/dconf, 4 ./.config/goa-1.0, 4 ./.config/update-notifier, 8 ./.config/gtk-3.0, 156 ./.config, 42892 ..

```
kb@phoenixNAP:~$ du | tail
12      ./.config/evolution
12      ./.config/ibus/bus
16      ./.config/ibus
84      ./.config/pulse
8       ./.config/dconf
4       ./.config/goa-1.0
4       ./.config/update-notifier
8       ./.config/gtk-3.0
156     ./.config
42892   .
kb@phoenixNAP:~$
```

Both **head** and **tail** commands are helpful when reading Linux log files.

19. diff command

The diff command (**d**ifference) compares two files and prints the difference. To use the command, run:

```
diff <file 1> <file 2>
```

For example, to compare files *test1.txt* and *test2.txt*, run:

```
diff file1.txt file2.txt
```

```
kb@phoenixNAP:~$ diff file1.txt file2.txt
1c1
< Hello world!
---
> Hello world
kb@phoenixNAP:~$
```

Developers often use **diff** to compare versions of the same code.

Note: Learn how to utilize `diff --color` to change the color of the output.

20. tar command

The tar command (**t**ape **a**rchiver) helps archive, compress, and extract archived files.

The command manages and creates files known as **tarballs**, which often appear during installation processes. The options provide different functionalities depending on the task.

21. chmod command

Use the **chmod** (**ch**ange **mo**de) command to change file and directory permissions. The command requires setting the permission code and the file or directory to which the permissions apply.

For example:

```
chmod <permission> <file or directory>
```

The permission is a number code consisting of three numbers:

- The first number is the permission of the current user (owner).
- The second number is the permission for the group.
- The third number is permissions for everyone else.

For example, to change the file permissions for a test.txt file so anyone can read, write, and execute, run:

```
chmod 777 file.txt
```

```
kb@phoenixNAP:~$ ls -l file.txt
-rw-rw-r-- 1 kb kb 0 cen 29 13:27 file.txt
kb@phoenixNAP:~$ chmod 777 file.txt
kb@phoenixNAP:~$ ls -l file.txt
-rwxrwxrwx 1 kb kb 0 cen 29 13:27 file.txt
```

Note: Allowing anyone to read, write, and execute files is considered a bad security practice. Implement privileged access management to maximize security on your system.

22. chown command

The **chown** command (**change ownership**) changes the ownership of a file or directory. To transfer ownership, use the following command as **sudo**:

```
sudo chown <new owner name or UID> <file or directory>
```

For example:

```
sudo chown bob file.txt
```

```
kb@phoenixNAP:~$ ls -l file.txt
-rwxrwxrwx 1 kb kb 0 cen 29 13:27 file.txt
kb@phoenixNAP:~$ sudo chown bob file.txt
kb@phoenixNAP:~$ ls -l file.txt
-rwxrwxrwx 1 bob kb 0 cen 29 13:27 file.txt
```

Configuring ownership is a common task during installations. The **chown** command allows daemons and processes to access files during setup.

23. ps command

The **ps** (process status) command lists processes currently running on the system. Every task creates a single or multiple processes running in the background.

Run **ps** without any options to see the running processes in the terminal session:

```
ps
```

```
kb@phoenixNAP:~$ ps
  PID TTY          TIME CMD
 1527 pts/0    00:00:00 bash
 15474 pts/0    00:00:00 ps
kb@phoenixNAP:~$
```

The output shows the process ID (PID), the terminal type, CPU time usage, and the command that started the process.

24. top command

The top command (**table of processes**) is an extended version of the **ps** command. Run the command without any options to see the result:

```
top
```

The output lists all running processes in real-time. To exit the viewer, press **CTRL+C**.

25. kill command

Use the **kill** command to terminate an unresponsive process. The command syntax is:

```
kill <signal option> <process ID>
```

There are sixty-four different signal numbers, but the most commonly used are:

- **-15** saves all progress before closing the process.
- **-9** forces a stop immediately.

The process ID (PID) is unique for every program. Use the **ps** or **top** command to find the PID of a process.

26. ping command

Use the ping command (**p**acket **i**nternet **g**roper) to check internet connectivity. The tool is valuable in troubleshooting networking issues. Add an address to test how it works, for example:

```
ping google.com
```

```
kb@phoenixNAP:~$ ping google.com
PING google.com (142.250.180.238) 56(84) bytes of data.
64 bytes from bud02s34-in-f14.1e100.net (142.250.180.238): icmp_seq=1 ttl=116 time=12.9 ms
64 bytes from bud02s34-in-f14.1e100.net (142.250.180.238): icmp_seq=2 ttl=116 time=14.0 ms
64 bytes from bud02s34-in-f14.1e100.net (142.250.180.238): icmp_seq=3 ttl=116 time=13.8 ms
64 bytes from bud02s34-in-f14.1e100.net (142.250.180.238): icmp_seq=4 ttl=116 time=14.7 ms
64 bytes from bud02s34-in-f14.1e100.net (142.250.180.238): icmp_seq=5 ttl=116 time=13.8 ms
64 bytes from bud02s34-in-f14.1e100.net (142.250.180.238): icmp_seq=6 ttl=116 time=14.1 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 12.941/13.898/14.659/0.512 ms
```


The output shows the response time from the website. Press **CTRL+C** to stop the ping. If no response shows, there's a problem connecting to the host.

27. wget command

The wget command (WWW **get**) is used to download files from the internet. Use the following syntax to download a file:

```
wget <URL>
```

The command is robust and can continue downloads in unstable and slow networks.

Note: Learn how to solve wget: command not found error.

28. uname command

Use the uname command (**U**nix **n**ame) to print system information. Add the **-a** option to print a complete overview:

```
uname -a
```

```
kb@phoenixNAP:~$ uname -a
Linux phoenixNAP 5.15.0-48-generic #54-Ubuntu SMP Fri Aug 26 13:26:29 UTC 2022
x86_64 x86_64 x86_64 GNU/Linux
kb@phoenixNAP:~$
```

The output shows the kernel version, OS, processor type, and other helpful information about the system.

29. history command

The terminal session keeps a history log of commands. To see the list, use the history command:

```
history
```

```
kb@phoenixNAP:~$ history
1  ./autorun.sh
2  reboot
3  vim
4  sudo apt-get install libncurses5-dev libncursesw5-dev
5  sudo apt install git
6  sudo apt install make
7  sudo apt install build-essentials
8  sudo apt install build-essential
9  sudo git clone https://github.com/vim/vim.git
10 cd vim/src
```

Add a number after the command to limit the number of entries if the list is long.

30. man command

The man command (**man**ual) is a convenient manual available in the terminal. Add **man** as a prefix to any command to check the manual reference:

```
man <command>
```

For example, to check the manual for the **man** command, run:

```
man man
```

```
MAN(1)                                Manual pager utils                                MAN(1)

NAME
    man - an interface to the system reference manuals

SYNOPSIS
    man [man options] [[section] page ...] ...
    man -k [apropos options] regex ...
    man -K [man options] [section] term ...
    man -f [whatis options] page ...
    man -l [man options] file ...
    man -w|-W [man options] page ...

DESCRIPTION
    man is the system's manual pager. Each page argument given to man is
    normally the name of a program, utility or function. The manual page
    associated with each of these arguments is then found and displayed.
```

To exit the manual, press **q**.

31. echo command

Use the echo command to print arguments to the terminal. The syntax is:

```
echo <argument>
```

For example, to print **Hello, world!** to the terminal run:

```
echo Hello, world!
```

```
kb@phoenixNAP:~$ echo Hello, world!  
Hello, world!  
kb@phoenixNAP:~$
```

The command helps append text to files, print program results, and display Linux environment variables.

32. hostname command

To check the DNS name of the current machine, use the hostname command:

```
hostname
```

```
kb@phoenixNAP:~$ hostname  
phoenixNAP  
kb@phoenixNAP:~$
```

The hostname shows in the terminal as a result. Advanced features include changing the hostname, viewing and changing the system's domain, and checking the IP address.

Result

The basics and status of Linux commands were studied completely.

Ex. No.: 5b

STUDY BASIC AND USER STATUS UNIX COMMANDS

AIM:

To study the basics of unix commands.

DEFINITIONS /THEORY:

Unix commands are a set of commands that are used to interact with the Unix operating system. Unix is a powerful, multi-user, multi-tasking operating system that was developed in the 1960s by Bell Labs. Unix commands are entered at the command prompt in a terminal window, and they allow users to perform a wide variety of tasks, such as managing files and directories, running processes, managing user accounts, and configuring network settings. Unix is now one of the most commonly used Operating systems used for various purposes such as Personal use, Servers, Smartphones, and many more. It was developed in the 1970's at AT & T Labs by two famous personalities Dennis M. Ritchie and Ken Thompson.

- You'll be surprised to know that the most popular programming language C came into existence to write the Unix Operating System.
- **Linux is Unix-Like operating system.**
- The most important part of the Linux is Linux Kernel which was first released in the early 90s by Linus Torvalds. There are several Linux distros available (most are open-source and free to download and use) such as Ubuntu, Debian, Fedora, Kali, Mint, Gentoo, Arch and much more.
- Now coming to the Basic and most usable commands of Linux/Unix part. (Please note that all the linux/unix commands are run in the terminal of a linux system. Terminal is like command prompt as that of in Windows OS)
- Linux/Unix commands are **case-sensitive** i.e Hello is different from hello.

Basic Unix commands:

- File System Navigation Unix Command
- File Manipulation Unix Command
- Process Management Unix Command
- Text Processing Unix Command
- Network Communication Unix Command
- Text Editors in Unix

File System Navigation Unix Command

Command	Description	Example
cd	Changes the current working directory.	cd Documents

Command	Description	Example
ls	Lists files and directories in the current directory.	ls
pwd	Prints the current working directory.	pwd
mkdir	Creates a new directory.	mkdir new_folder
rmdir	Removes an empty directory.	rmdir empty_folder
mv	Moves files or directories.	mv file1.txt Documents/

File Manipulation Unix Command

Command	Description	Example
touch	Creates an empty file or updates the access and modification times.	touch new_file.txt
cp	Copies files or directories.	cp file1.txt file2.txt
mv	Moves files or directories.	mv file1.txt Documents
rm	Remove files or directories.	rm old_file.txt
chmod	Changes the permissions of a file or directory.	chmod 644 file.txt
chown	Changes the owner and group of a file or directory.	chown user:group file.txt
ln	Creates links between files.	ln -s target_file symlink
cat	Concatenates files and displays their contents.	cat file1.txt file2.txt

Command	Description	Example
head	Displays the first few lines of a file.	head file.txt
tail	Displays the last few lines of a file.	tail file.txt
more	Displays the contents of a file page by page.	more file.txt
less	Displays the contents of a file with advanced navigation features.	less file.txt
diff	Compares files line by line.	diff file1.txt file2.txt
patch	Applies a diff file to update a target file.	patch file.txt < changes.diff

Process Management Unix Command

Command	Description	Example
ps	Displays information about active processes, including their status and IDs.	ps aux
top	Displays a dynamic real-time view of system processes and their resource usage.	top
kill	Terminates processes using their process IDs (PIDs).	kill <pid>
pkill	Sends signals to processes based on name or other attributes.	pkill -9 firefox

Command	Description	Example
killall	Terminates processes by name.	killall -9 firefox
renice	Changes the priority of running processes.	renice -n 10 <pid>
nice	Runs a command with modified scheduling priority.	nice -n 10 command
ps tree	Displays running processes as a tree.	ps tree
pgrep	Searches for processes by name or other attributes.	pgrep firefox
jobs	Lists active jobs and their status in the current shell session.	jobs
bg	Puts a job in the background.	bg <job_id>
fg	Brings a background job to the foreground.	fg <job_id>
nohup	Runs a command immune to hangups, with output to a specified file.	nohup command &
disown	Removes jobs from the shell's job table, allowing them to run independently.	disown <job_id>

Text Processing Unix Command

Command	Description	Example
---------	-------------	---------

Command	Description	Example
grep	Searches for patterns in text files.	grep "error" logfile.txt
sed	Processes and transforms text streams.	sed 's/old_string/new_string/g' file.txt
awk	Processes and analyzes text files using a pattern scanning and processing language.	awk '{print \$1, \$3}' data.csv

Network Communication Unix Command

Command	Description	Example
ping	Tests connectivity with another host using ICMP echo requests.	ping google.com
traceroute	Traces the route that packets take to reach a destination.	traceroute google.com
nslookup	Queries DNS servers for domain name resolution and IP address information.	nslookup google.com
dig	Performs DNS queries, providing detailed information about DNS records.	dig google.com
host	Performs DNS lookups, displaying domain name to IP address resolution.	host google.com

Command	Description	Example
whois	Retrieves information about domain registration and ownership.	whois google.com
ssh	Provides secure remote access to a system.	ssh username@hostname
scp	Securely copies files between hosts over a network.	scp file.txt username@hostname:/path/
ftp	Transfers files between hosts using the File Transfer Protocol (FTP).	ftp hostname
telnet	Establishes interactive text-based communication with a remote host.	telnet hostname
netstat	Displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.	netstat -tuln
ifconfig	Displays or configures network interfaces and their settings.	ifconfig
iwconfig	Configures wireless network interfaces.	iwconfig wlan0
route	Displays or modifies the IP routing table.	route -n

Command	Description	Example
arp	Displays or modifies the Address Resolution Protocol (ARP) cache.	arp -a
ss	Displays socket statistics.	ss -tuln
hostname	Displays or sets the system's hostname.	hostname
mtr	Combines the functionality of ping and traceroute, providing detailed network diagnostic information.	mtr google.com

System Administration Unix Command

Command	Description	Example
df	Displays disk space usage.	df -h
du	Displays disk usage of files and directories.	du -sh /path/to/directory
crontab -e	Manages cron jobs, which are scheduled tasks that run at predefined times or intervals.	crontab -e

Text Editors in Unix

Text Editor	Description	Example
Vi / Vim	Vi (Vim) is a highly configurable, powerful, and feature-rich text editor based on the original Vi editor. Vim offers modes for both command-line operations and text editing.	Open a file with Vim: vim filename Exit Vim editor: Press Esc, then type :wq and press Enter

Text Editor	Description	Example
Emacs	Emacs is a versatile text editor with extensive customization capabilities and support for various programming languages.	Open a file with Emacs: emacs filename Save and exit Emacs: Press Ctrl + X, then Ctrl + S and Ctrl + X, then Ctrl + C to exit
Nano	Nano is a simple and user-friendly text editor designed for ease of use and accessibility.	Open a file with Nano: nano filename Save and exit Nano: Press Ctrl + O, then Ctrl + X
Ed	Ed is a standard Unix text editor that operates in line-oriented mode, making it suitable for batch processing and automation tasks.	Open a file with Ed: ed filename Exit Ed editor: Type q and press Enter
Jed	Jed is a lightweight yet powerful text editor that provides an intuitive interface and support for various programming languages.	Open a file with Jed: jed filename Save and exit Jed: Press Alt + X, then type exit and press Enter

Inference:

Result

The basics and status of unix commands were studied completely.

Ex No: 6

Date:

**STUDY & USE OF COMMANDS FOR PERFORMING
ARITHMETIC OPERATIONS WITH UNIX/LINUX**

AIM

To Study & use of commands for performing arithmetic operations with Unix/Linux.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License). We can do arithmetic operations in shell script in a several way (using let command, using expr command).

PROGRAM

1. Using echo Command or Double Parenthesis

```
#!/bin/sh
```

```
a=10
```

```
b=20
```

```
val=`expr $a + $b`
```

```
echo "a + b : $val"
```

```
val=`expr $a - $b`
```

```
echo "a - b : $val"
```

```
val=`expr $a \* $b`  
echo "a * b : $val"
```

```
val=`expr $b / $a`  
echo "b / a : $val"
```

```
val=`expr $b % $a`  
echo "b % a : $val"
```

```
if [ $a == $b ]  
then  
    echo "a is equal to b"  
fi
```

```
if [ $a != $b ]  
then  
    echo "a is not equal to b"  
fi
```

OUTPUT

2. Using let Command

```
#!/bin/bash
```

```
x=10  
y=3
```

```
let "z = $(( x * y ))" # multiplication  
echo $z  
let z=$((x*y))  
echo $z
```

```
let "z = $(( x / y ))" # division
```

```
echo $z  
let z=$((x/y))  
echo $z
```

OUTPUT

3. Using expr Command

```
#!/bin/bash  
a=10  
b=3
```

```
# there must be spaces before/after the operator  
sum=`expr $a + $b`  
echo $sum
```

```
sub=`expr $a - $b`  
echo $sub
```

```
mul=`expr $a \* $b`  
echo $mul
```

```
div=`expr $a / $b`  
echo $div
```

OUTPUT

Pre and Post Lab Questions

1. What is the 'expr' command used for? Give an example.
2. How can you perform floating-point arithmetic in the shell?
3. Explain the difference between `$((...))` and `$(...)` in bash.
4. How do you use the 'bc' command for complex calculations?
5. What is the purpose of the 'scale' variable in 'bc'?
6. How can you generate random numbers in bash?
7. Explain how to use variables in arithmetic operations within a shell script.
8. What is the significance of the '\$?' variable after an arithmetic operation?
9. How do you perform exponentiation in bash?
10. Explain the use of the 'let' command for arithmetic operations.

RESULT:

Thus the Study & use of commands for performing arithmetic operations with Unix/Linux are Executed and the output is obtained successfully.

Ex No: 7

Date:

EXECUTE SHELL COMMANDS THROUGH VI EDITOR

AIM

To Execute shell commands through VI editor.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

The default editor that comes with the Linux/UNIX operating system is called vi (visual editor). Using vi editor, we can edit an existing file or create a new file from scratch. we can also use this editor to just read a text file. The advanced version of the vi editor is the vim editor.

Vi Command Mode :

When vi starts up, it is in Command Mode. This mode is where vi interprets any characters we type as commands and thus does not display them in the window. This mode allows us to move through a file, and delete, copy, or paste a piece of text. Enter into Command Mode from any other mode, requires pressing the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.

Vi Insert mode:

This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally, it is put in the file. The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode, you simply type

i. To get out of insert mode, press the Esc key, which will put you back into command mode.

Vi Last Line Mode (Escape Mode):

Line Mode is invoked by typing a colon [:], while vi is in Command Mode. The cursor will jump to the last line of the screen and vi will wait for a command. This mode enables you to perform tasks such as saving files and executing commands.

A shell is a special user program that provides an interface to the user to use operating system services. Shell accepts human-readable commands from the user and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

Vi COMMANDS

\$ vi <filename>	:	Open or edit a file
I	:	Switch to Insert mode
Esc	:	Switch to Command mode
:w	:	Save and continue editing
:wq or ZZ	:	Save and quit/exit vi
:q!	:	Quit vi and do not save changes
Yy	:	Yank (copy) a line of text
P	:	Paste a line of yanked text below the current line
o	:	Open a new line under the current line
O	:	Open a new line above the current line
A	:	Append to the end of the line
a	:	Append after the cursor's current position
I	:	Insert text at the beginning of the current line
b	:	Go to the beginning of the word
E	:	Go to the end of the word
x	:	Delete a single character
dd	:	Delete an entire line
Xdd	:	Delete X number of lines
Xyy	:	Yank X number of lines
G	:	Go to the last line in a file
XG	:	Go to line X in a file
gg	:	Go to the first line in a file
:num	:	Display the current line's line number
h	:	Move left one character
j	:	Move down one line
k	:	Move up one line
L	:	Move right one character

SHELL COMMANDS

1). Displaying the file contents on the terminal:

cat: It is generally used to concatenate the files. It gives the output on the standard output.

more: It is a filter for paging through text one screenful at a time.

less: It is used to viewing the files instead of opening the file. Similar to more command but it allows backward as well as forward movement.

head : Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.

tail : Used to print the last N-1 lines of a file. It accepts N as input and the default value of N is 10.

2). File and Directory Manipulation Commands:

mkdir : Used to create a directory if not already exist. It accepts the directory name as an input parameter.

cp : This command will copy the files and directories from the source path to the destination path. It can copy a file/directory with the new name to the destination path. It accepts the source file/directory and destination file/directory.

mv : Used to move the files or directories. This command's working is almost similar to cp command but it deletes a copy of the file or directory from the source path.

rm : Used to remove files or directories.

touch : Used to create or update a file.

3). Extract, sort, and filter data Commands:

grep : This command is used to search for the specified text in a file.

grep with Regular Expressions: Used to search for text using specific regular expressions in file.

sort : This command is used to sort the contents of files.

wc : Used to count the number of characters, words in a file.

cut : Used to cut a specified part of a file.

4). Basic Terminal Navigation Commands:

ls : To get the list of all the files or folders.

ls -l: Optional flags are added to ls to modify default behavior, listing contents in extended form -l is used for "long" output

ls -a: Lists of all files including the hidden files, add -a flag
cd: Used to change the directory.
du: Show disk usage.
pwd: Show the present working directory.
man: Used to show the manual of any command present in Linux.
rmdir: It is used to delete a directory if it is empty.
ln file1 file2: Creates a physical link.
ln -s file1 file2: Creates a symbolic link.
locate: It is used to locate a file in Linux System
echo: This command helps us move some data, usually text into a file.
df: It is used to see the available disk space in each of the partitions in your system.
tar: Used to work with tarballs (or files compressed in a tarball archive)

5). File Permissions Commands:

chown : Used to change the owner of the file.
chgrp : Used to change the group owner of the file.
chmod : Used to modify the access/permission of a user.

Pre and Post Lab Questions

1. How do you enter command mode in Vi to execute shell commands?
2. What is the syntax for executing a single shell command from within Vi?
3. How can you execute multiple shell commands in sequence from Vi?
4. Explain how to insert the output of a shell command into your current file in Vi.
5. What's the difference between using '!' and '!:!' for executing shell commands in Vi?
6. How do you use Vi to edit a command's output before inserting it into your file?
7. Can you execute a shell script from within Vi? If so, how?
8. How would you use a Vi command to sort the lines in your current file?
9. Explain how to use Vi to search and replace text using sed or awk commands.
10. How can you save the output of a shell command executed in Vi to a new file?

RESULT:

Thus the shell commands through VI editor are Executed and the output is obtained successfully.

AIM

To Study and use of the Command for changing file permissions.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

Using Linux as your operating system allows you to easily provide access to many users simultaneously. However, that access also presents potential security risks. Understanding the variety and types of Linux file permissions for users and groups will ensure that your system is optimally secure.

There are three options for permission groups available in Linux. These are

owners: these permissions will only apply to owners and will not affect other groups.

groups: you can assign a group of users specific permissions, which will only impact users within the group.

all users: these permissions will apply to all users, and as a result, they present the greatest security risk and should be assigned with caution.

There are three kinds of file permissions in Linux:

Read (r): Allows a user or group to view a file.

Write (w): Permits the user to write or modify a file or directory.

Execute (x): A user or group with execute permissions can execute a file or view a directory.

PERMISSION COMMANDS

1. Change directory permissions

chmod +rwx filename : to add permissions
chmod -rwx directoryname : to remove permissions.
chmod +x filename : to allow executable permissions.
chmod -wx filename : to take out write and executable permissions.

2. Change directory permissions for the Group Owners and Others

chmod g+w filename : to add permissions to group
chmod g-wx filename : to remove write and executable permissions to group.
chmod o+w filename : to allow write permissions to others.
chmod o-rwx foldername : to take out read, write and executable permissions to others.
chmod ugo+rwx foldername : to give read, write, and execute to everyone.
chmod a=r foldername : to give only read permission for everyone.

3. Change Groups of Files and Directories

chgrp groupname filename : To change the group ownership of a file or directory
chgrp groupname foldername : To change the group ownership of a folder

4. Changing ownership

chown name filename : Change ownership to file
chown name foldername : Change ownership to folder.
chown -R name:filename /home/name/directoryname : combine the group and ownership.

5. Changing Linux permissions in numeric code

0 = No Permission	+ Add permissions
1 = Execute	- Remove permissions
2 = Write	= Set the permissions to the specified values
3 = Write execute	u User
4 = Read	g Group
5 = Read execute	o Others
6 = Read write	a All three
7 = Read write execute	

Pre and Post Lab Questions

1. What is the primary command used to change file permissions in Unix/Linux?
2. Explain the meaning of the three sets of rwx in file permissions.
3. How do you use numeric (octal) notation to set file permissions?
4. What's the difference between chmod +x and chmod 755?
5. How can you recursively change permissions for a directory and its contents?
6. Explain the concept of SUID, SGID, and sticky bit. How do you set them?
7. What does the command "chmod go-rwx filename" do?
8. How do you view the current permissions of a file?
9. What's the difference between chmod and chown commands?
10. How would you give read and write permissions to the owner, and only read permissions to others?

RESULT:

Thus the Study and use of the Command for changing file permission are Executed and the output is obtained successfully.

Ex No: 9

Date:

DEVELOP SCHEDULING ALGORITHM FOR REAL TIME APPLICATIONS

AIM

To Develop scheduling algorithm for Real time Applications.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

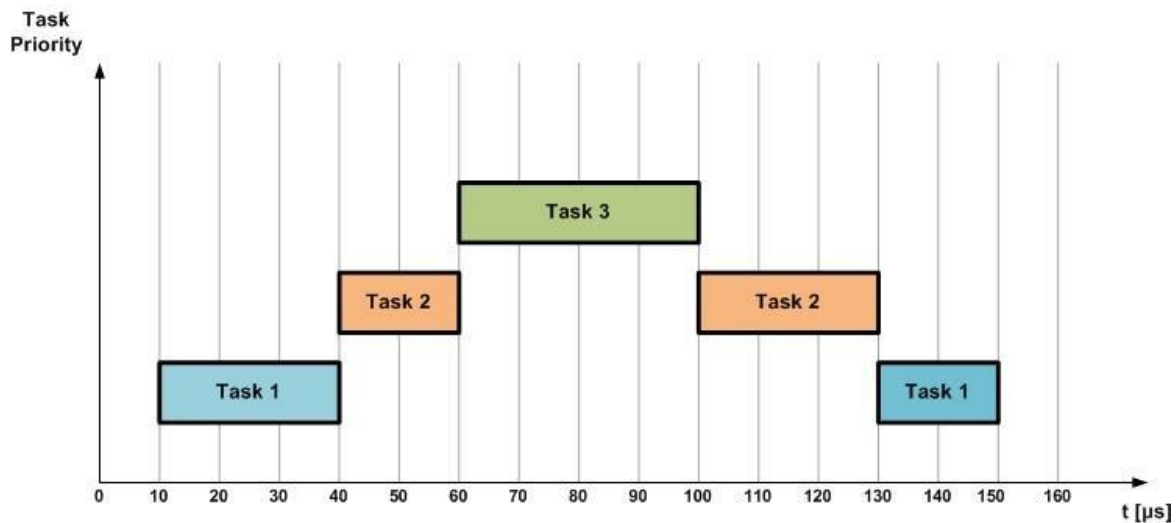
THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

There are many scheduling algorithms that can be used for scheduling task execution on a CPU. They can be classified into two main types: preemptive scheduling algorithms and non-preemptive scheduling algorithms.

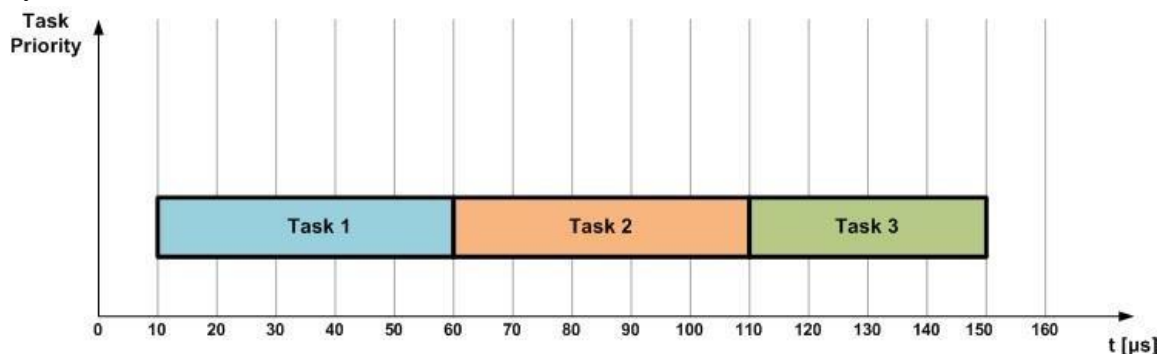
Preemptive Scheduling

Preemptive scheduling allows the interruption of a currently running task, so another one with more “urgent” status can be run. The interrupted task is involuntarily moved by the scheduler from running state to ready state. This dynamic switching between tasks that this algorithm employs is, in fact, a form of multitasking. It requires assigning a priority level for each task. A running task can be interrupted if a task with a higher priority enters the queue.



Non-preemptive Scheduling (a.k.a Co-Operative Scheduling)

In non-preemptive scheduling, the scheduler has more restricted control over the tasks. It can only start a task and then it has to wait for the task to finish or for the task to voluntarily return the control. A running task can't be stopped by the scheduler. The non-preemptive scheduling can simplify the synchronization of the tasks, but that is at the cost of increased response times to events. This reduces its practical use in complex real-time systems.



The most used algorithms in practical RTOS are non-preemptive scheduling, round-robin scheduling, and preemptive priority scheduling.

First Come, First Served (FCFS)

FCFS is a non-preemptive scheduling algorithm that has no priority levels assigned to the tasks. The task that arrives first into the scheduling queue (i.e enters ready state), gets put into the running state first and starts utilizing the CPU. It is a relatively simple scheduling algorithm where all the tasks will get executed eventually. The response time is high as this is a non-preemptive type of algorithm.

Shortest Job First (SJF)

In the shortest job first scheduling algorithm, the scheduler must obtain information about the execution time of each task and it then schedules the one with the shortest execution time to run next. SJF is a non-preemptive algorithm, but it also has a preemptive version. In the preemptive version of the algorithm (aka shortest remaining time) the parameter on

which the scheduling is based is the remaining execution time of a task. If a task is running it can be interrupted if another task with shorter remaining execution time enters the queue. A disadvantage of this algorithm is that it requires the total execution time of a task to be known before it is run.

Priority Scheduling

Priority scheduling is one of the most popular scheduling algorithms. Each task is assigned a priority level. The basic principle is that the task with the highest priority will be given the opportunity to use the CPU. In the preemptive version of the algorithm, a running task can be stopped if a higher priority task enters the scheduling queue. In the non-preemptive version of the algorithm once a task is started it can't be interrupted by a higher priority task. Of course, not all tasks can have unique priority levels and there will always be tasks that have the same priority. Different approaches can be used for handling the scheduling of those tasks (e.g FCFS scheduling or round-robin scheduling).

Round-Robin Scheduling

Round-robin is a preemptive type of scheduling algorithm. There are no priorities assigned to the tasks. Each task is put into a running state for a fixed predefined time. This time is commonly referred to as time-slice (aka quantum). A task can not run longer than the time-slice. In case a task has not completed by the end of its dedicated time-slice, it is interrupted, so the next task from the scheduling queue can be run in the following time slice. A preempted task has an opportunity to complete its operation once it's again its turn to use a time-slice. An advantage of this type of scheduling is its simplicity and relatively easy implementation.

PROGRAMS

First Come, First Served (FCFS)

```
#!/bin/bash sort() {
    context_switches=0
    for ((i = 0; i < $n; i++)); do
        for ((j = 0; j < `expr $n - $i - 1`; j++)); do
            if [ ${arrival_time[j]} -gt ${arrival_time[${j+1}]} ]; then
                # swap
                temp=${arrival_time[j]}
                arrival_time[j]=${arrival_time[${j+1}]}
                arrival_time[${j+1}]=$temp
                temp=${burst_time[j]}
                burst_time[j]=${burst_time [${j+1}]}
                burst_time[${j+1}]=$temp
                temp=${pid[j]}
                pid[j]=${pid[${j+1}]}
                pid[${j+1}]=$temp
            fi
            context_switches=$((context_switches+1))
        done
    done
}
```

```

        elif [ ${arrival_time[j]} -eq ${arrival_time[${(j+1)}]} ]; then
            if [ ${pid[j]} -eq ${pid[${(j+1)}]} ]; then
                temp=${arrival_time[j]}
            fi
        fi done
    done
}

arrival_time[$j]=${arrival_time[${(j+1)}]}
arrival_time[${(j+1)}]=$temp temp=${burst_time[j]}
burst_time[$j]=${burst_time[${(j+1)}]}
burst_time[${(j+1)}]=$temp temp=${pid[j]}
pid[$j]=${pid[${(j+1)}]} pid[${(j+1)}]=$temp
context_switches=$((context_switches+1))border(){ z=121
    for ((i=0; i<$z; i++))
    do
        echo -n " - "
    done
    echo ""
}

findWaitingTime(){
    service_time[0]=0
    waiting_time[0]=0
    for ((i=1; i<$n; i++))
    do
        z=1
        y=`expr $i - $z`
        service_time[$i]=`expr ${service_time[$y]} + ${burst_time[$y]} `
        waiting_time[$i]=`expr ${service_time[$i]} - ${arrival_time[$i]} `
        if [ ${waiting_time[$i]} -lt 0 ]
        then
            waiting_time[$i]=0
        fi
    done
}

findTurnAroundTime(){
    for ((i=0; i<$n; i++))
    do
        tat[$i]=`expr ${waiting_time[$i]} + ${burst_time[$i]} `
    done
}

findAverageTime() {
    sort

```

```

findWaitingTime
findTurnAroundTime
total_wt=0
total_tat=0
echo ""
border
printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time" "Arrival
time" "Waiting time" "Turn around time" "Completion time"
border
for ((i = 0; i < $n; i++)); do
    total_wt=$((total_wt + ${waiting_time[$i]}))
    total_tat=$((tat[$i] + total_tat))
    completion_time=$((arrival_time[$i] + tat[$i]))
    printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "${pid[$i]} ${burst_time[$i]}
${arrival_time[$i]} ${waiting_time[$i]} ${tat[$i]} $completion_time
done
border
echo ""
avgwt=`echo "scale=3; $total_wt / $n" | bc`
echo "Average waiting time = $avgwt"
avgtat=`echo "scale=3; $total_tat / $n" | bc`
echo "Average turn around time = $avgtat"
echo ""
echo "GANTT CHART: "
echo ""
for ((i = 0; i < 8 * n + n + 1; i++)); do
    echo -n "- "
done
echo ""

for ((i = 0; i < $n; i++)); do
    echo -n "| "
    echo -n "P${pid[$i]}"
    echo -n " "
done
echo "|"
for ((i = 0; i < 8 * n + n + 1; i++)); do
    echo -n "- "
done
echo ""
echo -n "0  "
for ((i = 0; i < $n; i++)); do
    echo -n "`expr ${arrival_time[$i]} + ${tat[$i]}`"

```

```

        echo -n "      "
    done
    echo ""
    echo "Number of context switches: $context_switches"
}

echo ""
echo "--FIRST COME FIRST SERVED SCHEDULING--"
echo ""
echo -n "Enter the number of processes: "
read n
for ((i = 0; i < $n; i++)); do
    echo -n "Enter Process Id: "
    read pid[$i]
    echo -n "Enter arrival time: "
    read arrival_time[$i]
    echo -n "Enter burst time: "
    read burst_time[$i]
done
findAverageTime

```

Shortest Job First (SJF)

```

#!/bin/bash
border() {
    z=121
    for ((i=0; i<$z; i++)); do
        echo -n "- "
    done
    echo ""
}

arrangeArrival() {
    z=1
    for ((i=0; i<$n; i++)); do
        for ((j=i+1; j<$n; j++)); do
            if [ ${arrival_time[$i]} -gt ${arrival_time[$j]} ]; then
                temp=${arrival_time[$j]}
                arrival_time[$j]=${arrival_time[$i]}
                arrival_time[$i]=$temp

                temp=${burst_time[$j]}
                burst_time[$j]=${burst_time[$i]}
                burst_time[$i]=$temp
            fi
        done
    done
}

```

```

        temp=${burst_time_copy[$j]}
        burst_time_copy[$j]=${burst_time_copy[$i]}
        burst_time_copy[$i]=$temp

        temp=${pid[$j]}
        pid[$j]=${pid[$i]}
        pid[$i]=$temp
    fi
done
done
}

arrangeBurst() {
    z=1
    for ((i=0; i<$n; i++)); do
        for ((j=i+1; j<$n; j++)); do
            if [ ${arrival_time[$i]} -eq ${arrival_time[$j]} ]; then
                if [ ${burst_time[$i]} -gt ${burst_time[$j]} ]; then
                    temp=${arrival_time[$j]}
                    arrival_time[$j]=${arrival_time[$i]}
                    arrival_time[$i]=$temp

                    temp=${burst_time[$j]}
                    burst_time[$j]=${burst_time[$i]}
                    burst_time[$i]=$temp

                    temp=${burst_time_copy[$j]}
                    burst_time_copy[$j]=${burst_time_copy[$i]}
                    burst_time_copy[$i]=$temp

                    temp=${pid[$j]}
                    pid[$j]=${pid[$i]}
                    pid[$i]=$temp
                fi
            fi
        done
    done
}

```

```

timecalc() { is_completed=0
current_time=0 cp=0
count=0
max=1000
context_switches=0
for ((i=0; i<$n; i++)); do
    if [ ${arrival_time[$i]} -le $current_time ]; then
        if [ ${burst_time[$i]} -lt $max ]; then
            if [ ${burst_time[$i]} -ne 0 ]; then
                cp=$i
                max=${burst_time[$i]}
                if [ ${burst_time[$i]} -eq ${burst_time_copy[$i]} ]; then
                    waiting_time[$i]=$current_time
                fi
            fi
        fi
    fi
done
while [ $is_completed -eq 0 ]; do
    if [ $count -eq $n ]; then
        is_completed=1
        h=$current_time
    fi
    chart[$current_time]=`expr $cp + 1`
    ((current_time++))
    if [ ${burst_time[$cp]} -gt 0 ]; then
        burst_time[$cp]=`expr ${burst_time[$cp]} - 1`
        max=${burst_time[$cp]}
        if [ ${burst_time[$cp]} -eq 0 ]; then
            ((count++))
            completion_time[$cp]=$current_time
            max=1000
        fi
    fi
    prevcp=$cp
    for ((i=0; i<$n; i++)); do
        if [ ${arrival_time[$i]} -le $current_time ]; then
            if [ ${burst_time[$i]} -lt $max ]; then
                if [ ${burst_time[$i]} -ne 0 ]; then
                    cp=$i
                    max=${burst_time[$i]}
                fi
            fi
        fi
    fi

```

```

        fi
    done
    if [ $prevcp -ne $cp ]; then
        waiting_time[$i]=$current_time
        context_switches=$((context_switches+1))
    fi
done

for ((i=0; i<$n; i++)); do
    waiting_time[$i]=`expr    ${completion_time[$i]}    -    ${arrival_time[$i]}    -
    ${burst_time_copy[$i]}`
    if [ ${waiting_time[$i]} -lt 0 ]; then
        waiting_time[$i]=0
    fi
    tat[$i]=`expr ${waiting_time[$i]} + ${burst_time_copy[$i]}`
done
total_wt=0
total_tat=0
echo ""
border
printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time" "Arrival
time" "Waiting time" "Turn around time" "Completion time"
border
for ((i=0; i<$n; i++)); do
    total_wt=`expr $total_wt + ${waiting_time[$i]}`
    total_tat=`expr ${tat[$i]} + $total_tat`
    completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
    printf          "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n"          ${pid[$i]}
    ${burst_time_copy[$i]}      ${arrival_time[$i]}      ${waiting_time[$i]}      ${tat[$i]}
    $completion_time
done
border
echo ""
avgwt=`echo "scale=3; $total_wt / $n" | bc`
echo "Average waiting time = $avgwt"
avgtat=`echo "scale=3; $total_tat / $n" | bc`
echo "Average turn around time = $avgtat"
echo ""
echo "GANTT CHART:"
echo ""
count_cols=1
cols_id[0]={chart[0]}
cols[0]=0

```

```

j=1
for ((i=1; i<$h; i++)); do
    if [ ${chart[$i]} -ne ${chart[`expr $i - 1`] } ]; then
        ((count_cols++))
        cols[$j]=$i
        cols_id[$j]=${chart[$i]}
        ((j++))
    fi
done
echo ""

for ((i=0; i<8*count_cols+count_cols+1; i++)); do
    echo -n "- "
done
echo ""

for ((i=0; i<$count_cols; i++)); do
    echo -n "| "
    echo -n "P${cols_id[$i]}"
    echo -n " "
done
echo "|"
for ((i=0; i<8*count_cols+count_cols+1; i++)); do
    echo -n "- "
done
echo ""
echo -n "0  "
for ((i=1; i<$count_cols; i++)); do
    echo -n "${cols[$i]}"
    echo -n "    "
done
echo -n "$h"
echo ""
echo "Number of context switches: $context_switches"
}
echo ""
echo "--PRE-EMPTIVE SHORTEST JOB FIRST SCHEDULING--"
echo ""
echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++)); do
    echo -n "Enter Process Id: "
    read pid[$i]

```

```

echo -n "Enter arrival time: "
read arrival_time[$i]
echo -n "Enter burst time: "
read burst_time[$i]
burst_time_copy[$i]={burst_time[$i]}
done
arrangeArrival
arrangeBurst
timecalc

```

Priority Scheduling

Function to implement Priority Scheduling Algorithm (non pre-emptive)

```

border(){
    z=121
    for ((i=0; i<$z; i++))
    do
        echo -n "- "
    done
    echo ""
}

arrangeArrival(){
    z=1
    for ((i=0; i<$n; i++))
    do
        for ((j=i+1; j<$n; j++))
        do
            if [ ${arrival_time[$i]} -gt ${arrival_time[$j]} ]
            then
                temp=${arrival_time[$j]}
                arrival_time[$j]=${arrival_time[$i]}
                arrival_time[$i]=$temp

                temp=${burst_time[$j]}
                burst_time[$j]=${burst_time[$i]}
                burst_time[$i]=$temp

                temp=${priority[$j]}
                priority[$j]=${priority[$i]}
                priority[$i]=$temp
            fi
        done
    done
}

```

```

        fi done
    done
}
temp=${pid[$j]} pid[$j]=${pid[$i]} pid[$i]=$temp

```

```

arrangePriority(){
    z=1
    for ((i=0; i<$n; i++))
    do
        for ((j=i+1; j<$n; j++))
        do
            if [ ${arrival_time[$i]} -eq ${arrival_time[$j]} ]
            then
                if [ ${priority[$i]} -gt ${priority[$j]} ]
                then
                    temp=${arrival_time[$j]}
                    arrival_time[$j]=${arrival_time[$i]}
                    arrival_time[$i]=$temp

                    temp=${burst_time[$j]}
                    burst_time[$j]=${burst_time[$i]}
                    burst_time[$i]=$temp

                    temp=${priority[$j]}
                    priority[$j]=${priority[$i]}
                    priority[$i]=$temp

                    temp=${pid[$j]}
                    pid[$j]=${pid[$i]}
                    pid[$i]=$temp
                fi
            fi
        done
    done
}

```

```

findWaitingTime() {
    service_time[0]=0
    waiting_time[0]=0

```

```

context_switches=0
current_process=${pid[0]}

for ((i=1; i<$n; i++)); do
    z=1
    y=`expr $i - $z`
    service_time[$i]=`expr ${service_time[$y]} + ${burst_time[$y]} `
    waiting_time[$i]=`expr ${service_time[$i]} - ${arrival_time[$i]} `
    if [ ${waiting_time[$i]} -lt 0 ]; then
        waiting_time[$i]=0
    fi
    # Check for context switch
    if [ "${pid[$i]}" != "$current_process" ]; then
        context_switches=$((context_switches+1))
        current_process=${pid[$i]}
    fi
done

echo "Number of context switches: $context_switches"
}

findTurnAroundTime(){
    for ((i=0; i<$n; i++))
    do
        tat[$i]=`expr ${waiting_time[$i]} + ${burst_time[$i]} `
    done
}

echo ""
echo "--NON PRE-EMPTIVE PRIORITY SCHEDULING--"
echo ""
echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++))
do
    echo -n "Enter Process Id: "
    read pid[$i]
    echo -n "Enter arrival time: "
    read arrival_time[$i]
    echo -n "Enter burst time: "
    read burst_time[$i]
    echo -n "Enter priority: "
    read priority[$i]

```

```

done
arrangeArrival
arrangePriority
findWaitingTime
findTurnAroundTime
total_wt=0
total_tat=0
echo ""
border
printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time" "Arrival
time" "Waiting time" "Turn around time" "Completion time"
border
for ((i=0; i<$n; i++))
do
    total_wt=`expr $total_wt + ${waiting_time[$i]}`
    total_tat=`expr ${tat[$i]} + $total_tat`
    completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
    printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]} ${burst_time[$i]}
${arrival_time[$i]} ${waiting_time[$i]} ${tat[$i]} $completion_time

    #echo "${burst_time[$i]}    ${arrival_time[$i]}    ${waiting_time[$i]}    ${tat[$i]}
$completion_time"
done
border
echo ""
avgwt=`echo "scale=3; $total_wt / $n" | bc`
echo "Average waiting time = $avgwt"
avgtat=`echo "scale=3; $total_tat / $n" | bc`
echo "Average turn around time = $avgtat"
echo ""

echo "GANTT CHART:"
echo ""
for ((i=0; i<8*n+n+1; i++))
do
    echo -n "- "
    done
    echo ""
for ((i=0; i<$n; i++))
do
    echo -n "| "
    echo -n "P${pid[$i]}"
    echo -n " "
done

```

```

echo "|"
for ((i=0; i<8*n+n+1; i++))
do
    echo -n "- "
done
echo ""
echo -n "0  "
for ((i=0; i<$n; i++))
do
    echo -n "`expr ${arrival_time[$i]} + ${tat[$i]}`      "
done
echo ""
rr(){
    # Function to implement Round Robin CPU scheduling algorithm.
sort() {
    for ((i = 0; i<$n; i++)); do
        for ((j = 0; j<`expr $n - $i - 1`; j++)); do
            if [ ${arrival_time[j]} -gt ${arrival_time[$((j+1))]} ]; then
                # swap
                temp=${arrival_time[j]}
                arrival_time[$j]=${arrival_time[$((j+1))]}

                arrival_time[$((j+1))]=$temp

                temp=${burst_time[j]}
                burst_time[$j]=${burst_time[$((j+1))]}
                burst_time[$((j+1))]=$temp

                temp=${pid[j]}
                pid[$j]=${pid[$((j+1))]}
                pid[$((j+1))]=$temp

                temp=${burst_time_copy[j]}
                burst_time_copy[$j]=${burst_time_copy[$((j+1))]}
                burst_time_copy[$((j+1))]=$temp

                temp=${arrival_time_copy[j]}
                arrival_time_copy[$j]=${arrival_time_copy[$((j+1))]}
                arrival_time_copy[$((j+1))]=$temp
            elif [ ${arrival_time[j]} -eq ${arrival_time[$((j+1))]} ]; then
                if [ ${pid[j]} -eq ${pid[$((j+1))]} ]; then
                    temp=${arrival_time[j]}
                    arrival_time[$j]=${arrival_time[$((j+1))]}

```

```
arrival_time[$((j+1))]=$temp
```

```
temp=${burst_time[j]}
```

```
burst_time[$j]=${burst_time[$((j+1))]}
```

```
burst_time[$((j+1))]=$temp
```

```
temp=${pid[j]}
```

```
pid[$j]=${pid[$((j+1))]}
```

```
pid[$((j+1))]=$temp
```

```
temp=${burst_time_copy[j]}
```

```
burst_time_copy[$j]=${burst_time_copy[$((j+1))]}
```

```
burst_time_copy[$((j+1))]=$temp
```

```
fi
```

```
fi
```

```
done
```

```
done
```

```
}
```

```
}
```

```
temp=${arrival_time_copy[j]}
```

```
arrival_time_copy[$j]=${
```

```
arrival_time_copy[$((j
```

```
+1))]}
```

```
arrival_time_copy[$((j
```

```
+1))]=$temp
```

Round-Robin Scheduling

```
#!/bin/bash
cnt=0
j=0
n=0
t=0
remain=0
flag=0
tq=0
wt=0
tat=0
context_switches=0
declare -a at bt rt
echo -n "Enter Total Process: "
read n
remain=$n
for ((cnt = 0; cnt < n; cnt++)); do
    echo -n "Enter Arrival Time and Burst Time for Process Process Number $((cnt + 1)): "
    read at[cnt]
    read bt[cnt]
    rt[cnt]={bt[cnt]}
done

echo -n "Enter Time Quantum: "
read tq
echo -e "\n\nProcess\t|Turnaround Time|Waiting Time\n\n"
for ((t = 0, cnt = 0; remain != 0; )); do
    if ((rt[cnt] <= tq && rt[cnt] > 0)); then
        t=$((t + rt[cnt]))
        rt[cnt]=0
        flag=1
    elif ((rt[cnt] > 0)); then
        rt[cnt]=$((rt[cnt] - tq))
        t=$((t + tq))
        ((context_switches++))
    fi
    if ((rt[cnt] == 0 && flag == 1)); then
        remain=$((remain - 1))
        echo "P$((cnt + 1))" "$t\t" "$((t - at[cnt]))" "$t\t" "$((t - at[cnt] - bt[cnt]))"
        wt=$((wt + t - at[cnt] - bt[cnt]))
        tat=$((tat + t - at[cnt]))
        flag=0
    fi
done
```

```

if ((cnt == n - 1)); then

    cnt=0
elif ((at[cnt + 1] <= t)); then
    cnt=$((cnt + 1))
else
    cnt=0
fi
done
echo -e "\nAverage Waiting Time= $(awk 'BEGIN {printf "%.2f", '$wt'/'$n'}')'"
echo -e "\nAvg Turnaround Time = $(awk 'BEGIN {printf "%.2f", '$tat'/'$n'}')'"
echo -e "\nNumber of Context Switches: $context_switches"
exit 0

```

Pre and Post Lab Questions

1. What are the real-time scheduling policies available in Linux, and how do they differ from standard scheduling policies?
2. Explain the purpose and usage of the SCHED_FIFO and SCHED_RR scheduling classes in Linux.
3. How can you set and modify the priority of a real-time process in Linux using system calls?
4. What is the role of the Linux kernel's completely fair scheduler (CFS) in real-time applications?
5. How does the PREEMPT_RT patch modify Linux for improved real-time performance?
6. Explain how to use the chrt command to set real-time attributes for a process in Linux.
7. What are the limitations of using standard Linux for hard real-time applications?
8. How can you implement periodic tasks in Linux using timer functions like timer_create()?
9. Explain the concept of CPU affinity and how it can be used to improve real-time performance in multi-core systems.
10. How can you measure and analyze the scheduling behavior of real-time tasks in Linux using tools like ftrace or perf?

RESULT:

Thus the programs for scheduling algorithm for Real time Applications are Executed and the output is obtained successfully.

AIM

To Develop a Mini project using Embedded Linux Platform.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

TENTATIVE PROJECT TITLES

1. Smart Home Automation System
 - Control lights, temperature, and appliances
 - Implement voice control or smartphone app interface
 2. Weather Station
 - Collect temperature, humidity, and pressure data
 - Display on LCD and upload to a web server
 3. Network Attached Storage (NAS) Device
 - Create a simple file server
 - Implement user authentication and file sharing
 4. Surveillance Camera System
-

- Capture video and stream over network
 - Implement motion detection and alert system
5. Digital Signage Player
 - Display dynamic content on a screen
 - Support remote content updates
 6. Raspberry Pi-based Robot
 - Implement movement control and sensor integration
 - Add computer vision capabilities
 7. IoT Gateway
 - Collect data from various sensors
 - Process and forward data to cloud services
 8. Embedded Music Player
 - Play audio files from local storage or streaming services
 - Implement playlist management and equalizer
 9. Vehicle Tracking System
 - Use GPS module for location tracking
 - Transmit data over cellular network
 10. Smart Mirror
 - Display time, weather, and personalized information
 - Implement gesture or voice control
 11. Industrial Process Monitor
 - Collect and display data from industrial sensors
 - Implement alarm system for out-of-range values
 12. Automatic Plant Watering System
 - Monitor soil moisture and control water pump
 - Implement scheduling and remote monitoring
 13. Energy Monitoring System
 - Measure and log power consumption
 - Provide analytics and suggestions for energy saving
 14. Facial Recognition Door Lock
 - Use camera and ML for face recognition
 - Control electronic door lock
 15. Portable Game Console
 - Emulate retro games
 - Implement custom controller interface
-

Pre and Post Lab Questions

1. What embedded Linux platform did you choose for your project, and why?
2. Explain the boot process of your embedded Linux system. How does it differ from a standard PC boot process?
3. How did you cross-compile your application for the target embedded platform?
4. What challenges did you face in integrating hardware peripherals with your embedded Linux system?
5. Explain how you implemented real-time features in your project using Linux. Did you use any specific scheduling policies?
6. How did you manage power consumption in your embedded Linux project?
7. What method did you use for inter-process communication in your project? Why did you choose this method?
8. Explain how you implemented error handling and system recovery in your embedded Linux application.
9. How did you ensure the security of your embedded Linux system, especially if it's network-connected?
10. What tools and techniques did you use for debugging your application on the embedded Linux platform?

RESULT:

Thus the Mini project using Embedded Linux Platform is completed and the output is obtained successfully.
